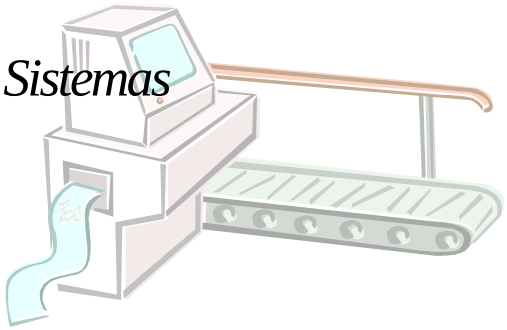
Eliseo Garcia García
e-mail: eliseo.garcia@uah.es
Despacho: N-245
Departamento de Automática

Manejo de GNU/Linux

Desarrollo de Aplicaciones para Sistemas Industriales



GNU/Linux y los sistemas empotrados

Introducción

Los Sistemas Operativos de Tiempo Real (SS.OO.T.R.) están teniendo una gran auge. Su principal campo de aplicación son las tareas de control complejas algunas de las cuales empiezan a integrar elementos característicos de la informática no industrial. Estas necesitan echar mano de recursos computacionales y de almacenamiento mucho más elevados que los que necesitaría una aplicación de control digital clásico por separado. La integración de estas dos vertientes de la informática resulta muy atractiva ya que permite generar soluciones eficaces y económicas.

Basándose en el S.O. de fuentes abiertas GNU/Linux se ha creado una variante llamada RTLinux, que logra incorporar extensiones de Tiempo Real al núcleo del sistema. Mediante RTLinux se pueden implementar los llamados Sistemas de Tiempo Real (S.T.R.) duros o estrictos. Esto quiere decir, que podemos tener un conjunto de tareas funcionando de una forma totalmente determinista, por lo que se podrá conocer a priori los tiempos de respuesta ante los eventos de entrada.

RTLinux es por tanto un sistema híbrido ya que por otra parte ofrece total compatibilidad con los programas desarrollados para GNU/Linux, ejecutándolos como tareas sin restricciones de Tiempo Real.

Ya que se van a desarrollar programas para RTLinux, será necesario tener un conocimiento del manejo básico de GNU/Linux.

En ese documento veremos la interacción básica con el intérprete de órdenes, y algunas de las órdenes y conceptos necesarios para el trabajo a nivel de usuario. En ningún caso la información aquí contenida pretende ser una guía completa de aprendizaje de GNU/Linux, sino más bien un resumen de algunos de los conceptos fundamentales que serán necesarios para que más adelante se pueda realizar una programación de sistema efectiva. Por tanto aunque no es estrictamente necesario, es recomendable que el alumno cuente con algún otro tipo de referencia más completa para afrontar los primeros pasos de contacto con el sistema. Así mismo sería beneficioso que el alumno se familiarizase con el sistema de ayuda del propio S.O. que aunque resulte en ocasiones un tanto críptico, es el más explícito y actualizado.

Nociones previas

Cuando se habla de GNU/Linux, a menudo nos referimos a un S.O. de propósito general tipo Unix que está construido alrededor del núcleo (*kernel*) de S.O. ideado originalmente por Linus Torvalds. Estrictamente hablando Linux hace referencia tan solo al núcleo del sistema, es decir, la parte característica de más bajo nivel de cualquier S.O.. El resto de aplicaciones son complementos realizados por terceras partes.

Para la realización fundamental del sistema, se emplean programas del proyecto GNU que persigue dotar a los desarrolladores de herramientas avanzadas que cuentan con código fuente abierto, para la realización de SS.OO.. GNU es un acrónimo recursivo de *GNU is Not Unix*; sus aplicaciones se han portado a diversas plataformas hardware, y a su vez están disponibles para una gran cantidad de SS.OO. no vinculados necesariamente con el mundo Unix, como bien indica su nombre. Las herramientas GNU respetan en gran medida los estándares POSIX[1], ANSI, e ISO, al igual que otras normalizaciones y recomendaciones de menor envergadura. Dado que para la creación de un S.O. completo la importancia de

[1] La compatibilidad del estándar POSIX que ofrece GNU/Linux está presente tanto en el los comandos que permiten el manejo del S.O. como en las bibliotecas de programación de C, y en otras características de bajo nivel.

GNU es como mínimo igual que la del propio Linux, se recomienda que se nombren conjuntamente como GNU/Linux para referirse genéricamente a los SS.OO. desarrollados en torno a ambos. Por convenio, utilizaremos la denominación Linux para referirnos exclusivamente al núcleo del sistema.

Primeros pasos

Dado que Linux nace como un clon de Unix, hereda muchos conceptos de este, algunos de los cuales los iremos viendo a continuación.

Linux es un S.O. multiusuario por lo que ofrece la posibilidad de que coexistan distintos usuarios conectados a la misma máquina. Estos serán atendidos de forma independiente.

Como consecuencia fundamental el sistema solicita el nombre y la clave a cualquier usuario que pretenda acceder al mismo. A continuación se muestra la pantalla de inicio de un sistema GNU/Linux basado en la distribución Red Hat versión 6.1:

```
Red Hat Linux release 6.1 (Cartman)
Kernel 2.2.18-rtl on an i586
login: david
Password:
Last login: Sun Jul 21 23:37:13 from P5.CASA
[david@P5 david]$
```

La primera línea indica el nombre del S.O. junto con su número de versión. La segunda indica la versión del núcleo y la máquina sobre la que se está ejecutando. En este caso, vemos que se trata de un núcleo versión `2.2.18-rtl` compilado para un micro de tipo Pentium. La coletilla `rtl` significa que se trata de un núcleo modificado al que se le han añadido las extensiones de Tiempo Real.

Una vez que se nos han dado estos datos fundamentales, la máquina se queda esperando a la identificación del usuario, en este caso `david`. Seguidamente pide la clave para asegurarse de que no se trata de un impostor. Una vez aceptada, se muestra un mensaje que indica la última vez que entró este usuario en el sistema, y la máquina desde donde se conectó, en este caso fue desde un terminal llamado `P5` con el dominio `CASA`.

Finalmente se nos muestra el *inductor de órdenes* o *shell* (concha) desde el que podremos introducir las órdenes oportunas. El inductor es mostrado por una aplicación llamada *intérprete de órdenes*, que nos va a servir de interfaz entre nuestra consola formada esencialmente por teclado y monitor, y el resto del sistema. En estas prácticas vamos a considerar que el intérprete sea *bash*, aunque podríamos elegir cualquier otro. El principal propósito de los intérpretes de órdenes es la de poder iniciar otros programas, aunque disponen de otras muchas funciones orientadas a facilitar el trabajo con el sistema.

En este caso el inductor muestra el nombre de usuario, y el nombre de la máquina a la que pertenece, seguido del nombre del directorio donde se encuentra. Vemos que el usuario `david` perteneciente a la máquina `P5`, y se encuentra en el directorio `david`. Es normal que para cada usuario que esté dado de alta en el sistema exista un directorio con el mismo nombre donde pueda almacenar sus documentos. El símbolo `$` (dólar) que aparece al final es de especial interés. Significa que se trata de un usuario *normal* del sistema. La información que aparece previa al símbolo es configurable, por lo que no todos los sistemas tipo Unix muestran el mismo contenido. En lo sucesivo por brevedad, el inductor tan sólo se mostrará con este tipo de carácter. Lo que introduzcamos a continuación y hasta el carácter de nueva línea, generado al presionar la tecla [Enter], lo llamaremos *línea de comando*, y habitualmente serán órdenes para el sistema. Podemos insertar comentarios en una línea de órdenes anteponiéndoles el carácter #.

El hecho de que se trate de un sistema multiusuario conlleva múltiples implicaciones del diseño: el sistema de ficheros permite asignar distintos permisos a los archivos de disco para preservar la privacidad entre usuarios así como la seguridad e integridad de los mismos. Existe un usuario especial con el nombre prefijado de *root*, al que a menudo se le suele llamar también *súper usuario*, porque tiene privilegios de acceso a todo el sistema. Generalmente, las aplicaciones en Unix "recuerdan" al usuario que las creó, y heredan sus permisos de acceso a archivos. Cualquier aplicación que haya sido invocada incorrectamente o bien que tenga propósitos maliciosos que haya sido ejecutada directa o indirectamente por el *root*, podría causar resultados desastrosos. Es por ello recomendable utilizar al usuario *root* lo menos posible, ya que así existirá una probabilidad menor de hacer daño al sistema o a otros usuarios. Cuando se ejecute el *shell* con privilegios de *root*, mostrará un símbolo `#` en el inductor donde normalmente mostraría `$`, lo que sirve para recordarnos que actuemos con cautela en la introducción de órdenes.

Al igual que en otros sistemas Unix, en GNU/Linux se utilizan las consolas de texto como medio estándar de trabajo. Aunque acostumbrados a las modernas interfaces gráficas, la interacción con el S.O. a través de una consola de texto puede resultar arcaica, tiene numerosas ventajas. A través de una consola podemos trabajar con órdenes estándar con muchos SS.OO. tipo Unix, incluso versiones antiguas; podemos contar con que las órdenes clásicas seguirán estando ahí, no teniendo que reaprender el manejo del sistema con cada nuevo entorno de escritorio que aparezca en el mercado. Por otra parte, la simplicidad en el intercambio de información que ofrecen las consolas de texto puede ser útil para conectar un sistema por grande y complejo que sea, a un discreto terminal como pudiera ser un microcontrolador conectado al bus serie RS-232 del ordenador, que podría perfectamente ser visto por el sistema como un usuario más.

En las consolas de texto se utilizan los conceptos de *entrada estándar*, *salida estándar*, y *salida estándar de errores*. La entrada y salida estándar son la fuente y el destino de datos que tiene una aplicación por defecto, que son habitualmente, el teclado y la pantalla, respectivamente. La salida estándar de errores es el destino de los posibles mensajes de error que pudieran generar los programas en ejecución, suelen ser dirigidos al mismo sitio donde se presente la salida estándar. A menudo se habla de *flujos* de entrada y salida de datos. El S.O. hace que estos tres flujos de datos por defecto sean vistos por las aplicaciones como archivos desde donde se puede leer o escribir información. Más adelante veremos cómo se puede alterar el funcionamiento normal que las aplicaciones hacen de los mismos.

Referencia de órdenes

Veamos ahora algunas de las órdenes estándares de las que disponemos:

pwd

Program Working Directory

Al igual que ocurre con otros SS.OO. con soporte para archivos de disco, en GNU/Linux existe el concepto de *directorio actual*. Se trata de una cadena de texto que contiene una *ruta absoluta* dentro del sistema de directorios. La utilizan las aplicaciones para confeccionar rutas válidas de archivo en el caso de requerir acceder a uno del que no se hubiera proporcionado la ruta completa. Para saber el directorio actual del intérprete de órdenes, usamos `pwd`:

```
$ pwd
/home/David
```

Vemos que la ruta por defecto es `/home/David`. Dentro del directorio `home` se almacenan las carpetas personales de los usuarios de la máquina. El carácter `/` se utiliza para separar nombres de directorios, excepto el primero, que indica el llamado *directorio raíz* del cual "cuelgan" el resto de directorios[2] y archivos. No confundir el directorio raíz (*root*) con el usuario *root*.

Cada nuevo proceso heredará el directorio actual de su padre en el momento de su creación. Este podrá ser variado durante la vida del mismo.

[2] En los sistemas de archivos que utiliza habitualmente Linux (ext2, ext3, …) un directorio es un caso particular de un archivo que se utiliza principalmente para almacenar información de los nombres de otros archivos, y las posiciones que les corresponde en el medio físico. De esta forma se crea un sistema de indexación de información útil para clasificar y almacenar datos.

Cambia el directorio actual que mantenía el intérprete de órdenes. A continuación se muestran algunos ejemplos del uso de `cd`:

Cambiar al directorio raíz:

```
$ cd /
```

Cambiar al directorio de usuarios `/home`:

```
$ cd /home
```

En estos dos ejemplos se han utilizado rutas absolutas del sistema de archivos. Si en este punto quisiéramos colocarnos de nuevo en el directorio `/home/David`, podríamos proporcionar igualmente la ruta absoluta, o bien indicar tan solo la parte de la ruta desde el directorio actual:

```
$ pwd
/home
$ cd David
$ pwd
/home/David
```

Apréciese que en UNIX se hace diferenciación entre caracteres en mayúscula y en minúscula en los nombres de archivo, considerándose diferentes dos archivos que se llamaran `notas.txt` y `Notas.txt`.

Podemos volver desde cualquier ubicación a nuestro directorio personal introduciendo `cd` sin argumentos:

```
$ pwd
/usr/src
$ cd
$ pwd
/home/David
```

También podemos utilizar el carácter tilde ~ como sustitutivo de la ruta absoluta de nuestro directorio de usuario. Podemos obtener dicho carácter mediante la pulsación de [AltGr]+[4]. Suponiendo que existe el directorio `/home/David/GNUstep` al que quisiéramos acceder teniendo configurado cualquier directorio de trabajo:

```
$ cd ~/GNUstep
$ pwd
/home/David/GNUstep
```

Observamos que podemos identificar fácilmente una ruta absoluta si comienza por / o por ~.

En cada directorio existen dos nombres de directorios adicionales válidos para formar rutas: el punto que hace referencia a ese mismo directorio, y los dos puntos seguidos, que hacen referencia al directorio previo en la ruta absoluta.

```
$ pwd
/home/David
$ cd ..
$ pwd
/home
$ cd ./David/././GNUstep/../GNUstep/./../..
$ pwd
/home
```

ls — List

Se utiliza para listar el contenido del directorio donde nos encontramos en ese momento. Si se introduce tal cual, muestra los nombres de los archivos visibles que se encuentren en ese directorio. Si `ls` no muestra nombres de archivos es porque ese directorio no contiene archivos visibles. En un sistema Unix los propios directorios son realmente archivos cuyo objetivo es almacenar los nombres de otros archivos que decimos que contiene. `ls` en principio no hace diferenciación entre directorios y otro tipo de archivos.

```
$ ls
GNUstep  startx notas.txt
```

En este caso nos dice que en el directorio actual existen tres archivos llamados `GNUstep, startx, y notas.txt`.

La inmensa mayoría de las órdenes de Unix disponen de modificadores opcionales que permiten especificar un comportamiento específico para la orden. En el caso de `ls`, si quisiéramos un listado detallado línea a línea, utilizaríamos el modificador `l`; si quisiéramos que mostrara información de todos los archivos contenidos, usaríamos el modificador `a`. Los modificadores se escriben seguidos al nombre de la orden, y precedidos por un guión:

```
$ ls -l -a
total 2
drwxr-xr-x    4 David    Ninguno         0 Jul 24 02:01 .
drwxr-xr-x    3 David    Ninguno         0 Jul 18 12:22 ..
-rw-r--r--    1 David    Ninguno       741 Aug  5 17:38 .bash_history
drwxr-xr-x    2 David    Ninguno         0 Jul 21 23:13 .fvwm
drwxr-xr-x    5 David    Ninguno         0 Jul 18 14:28 GNUstep
-rw-r--r--    1 David    Ninguno       324 Aug  8 15:31 notas.txt
lrwxrwxrwx    1 David    Ninguno       143 Jul 24 02:01 startx -> /usr/X11R6/bin
/startxwin.sh
```

Aparece la información solicitada en formato tabulado por columnas.

En la columna de la izquierda aparece la información relativa a la naturaleza del archivo, y los permisos de acceso: Si comienza por una `d` significa que esa línea hace referencia a un directorio, si comienza por una `l`, es que se trata de un enlace a otro archivo, y si comienza por un guión `-`, es que se trata de un archivo de datos ordinario. Existen más tipos de archivo que se irán citando según sea necesario. Después de este primer carácter, aparecen tres campos consecutivos de tres caracteres cada uno que en el caso de archivos normales indican los permisos de lectura `r`, escritura `w`, y ejecución `x` del archivo correspondientes al propietario, al grupo, y al resto de usuarios, respectivamente. Si la letra correspondiente aparece sustituida por un guión, significa que ese permiso está deshabilitado.

En el caso de tratarse de directorios, los permisos tienen significados distintos:

- `r` significa que podemos listar el contenido del directorio.
- `w` indica que se pueden añadir o borrar ficheros del directorio.

- `x` significa que podemos consultar información de los archivos del directorio, es decir listar el directorio. Los permisos de lectura y ejecución de directorios se suelen asignar al mismo tiempo.

Las siguientes columnas muestran el número de referencias que tiene ese archivo, el nombre del propietario, el nombre del grupo al que pertenece el propietario, la cantidad de *bytes* útiles que ocupa en el disco, la fecha de su última modificación, y el nombre.

Es posible combinar los modificadores de un solo carácter poniéndolos contiguos precedidos únicamente de un guión, por lo que la siguiente orden tendrá el mismo efecto:

```
$ ls -la
...
```

Como podemos ver, si no empleamos el modificador `a`, no aparecen los archivos cuyos nombres comienzan por un punto. Es un convenio el que `ls` considere a estos archivos como ocultos. Los archivos ocultos suelen ser archivos de texto que utilizan algunas aplicaciones para almacenar sus configuraciones en el disco duro, aunque cualquier usuario puede también usarlos según su conveniencia.

Un modificador muy útil que incorporan casi todas las órdenes es `help`. Sirve para pedir una breve descripción del propósito de esa orden, así como de todos los modificadores que admita. Los modificadores que ocupan varios caracteres deben ir precedidos por dos guiones consecutivos `--` en lugar de uno solo:

```
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuSUX nor --sort.

  -a, --all                  do not hide entries starting with .
  -A, --almost-all           do not list implied . and ..

. . .

  -x                         list entries by lines instead of by columns
  -X                         sort alphabetically by entry extension
  -1                         list one file per line
      --help                 display this help and exit
      --version              output version information and exit

By default, color is not used to distinguish types of files.  That is
equivalent to using --color=none.  Using the --color option without the
optional WHEN argument is equivalent to using --color=always.  With
--color=auto, color codes are output only if standard output is connected
to a terminal (tty).
```

Como vemos se hace distinción entre modificadores escritos en mayúsculas y en minúsculas.

Hasta el momento se ha listado el contenido del directorio de trabajo que tuviera el intérprete de órdenes en el momento de invocar a `ls`. Si quisiéramos ver el contenido de un directorio distinto al que nos encontramos, puede ser especificado como último parámetro de la orden. A continuación se muestra el contenido del directorio `GNUstep`, para lo cual sólo se indica su ruta relativa.

```
$ ls -l GNUstep
total 0
drwxr-xr-x    2 David    Ninguno         0 Jul 21 19:03 Defaults
drwxr-xr-x    4 David    Ninguno         0 Jul 18 14:28 Library
```

Como vemos contiene a su vez dos directorios.

Si quisiéramos mostrar el contenido de cualquiera de ellos tan solo deberíamos indicarlo posponiéndolo al nombre de su directorio padre separándolos con un carácter de dividir / aun contando con una ruta relativa:

```
$ ls -l GNUstep/Library
total 0
drwxr-xr-x    2 David    Ninguno         0 Jul 18 14:28 Icons
drwxr-xr-x    9 David    Ninguno         0 Jul 21 19:02 WindowMaker
```

Para listar el contenido del directorio raíz, haríamos lo siguiente:

```
$ ls -l /
drwxr-xr-x   2 root     root         4096 Jul  7 15:55 bin
drwxr-xr-x   2 root     root         4096 Jul 21 18:12 boot
drwxr-xr-x   6 root     root        36864 Aug  8 18:37 dev
drwxr-xr-x  30 root     root         4096 Aug  8 18:37 etc
drwxr-xr-x   7 root     root         4096 Jul  7 14:31 home
drwxr-xr-x   4 root     root         4096 Jul  7 15:42 lib
drwxr-xr-x   2 root     root        16384 Jul  7 15:23 lost+found
drwxr-xr-x  10 root     root         4096 Jul 13 16:37 mnt
drwxr-xr-x   3 root     root         4096 Jul 13 19:37 opt
dr-xr-xr-x  50 root     root            0 Aug  8 20:36 proc
drwxr-x---  18 root     root         4096 Jul 21 23:37 root
drwxr-xr-x   3 root     root         4096 Jul  7 15:57 sbin
drwxrwxrwt   9 root     root         4096 Aug  8 19:41 tmp
drwxr-xr-x  20 root     root         4096 Jul 12 16:34 usr
drwxr-xr-x  18 root     root         4096 Jul  7 15:57 var
```

Aquí podemos ver algunos de los directorios clásicos en todo sistema Unix, cuyos contenidos son los siguientes:

- **/bin** y **/usr/bin**.- Programas ejecutables del sistema como las órdenes del *shell*.
- **/boot**.- Núcleos del S.O. que se cargan al iniciar el sistema, más otros archivos de configuración iniciales.
- **/dev** .- Referencias a nombres lógicos de dispositivos a través de los cuales podremos interactuar con el hardware. No todas las entradas en este directorio se corresponden con dispositivos físicos existentes.
- **/etc** y **/mnt**.- Archivos y directorios usados por subsistemas de archivos como unidades de red, u otras particiones adicionales.
- **/home**.- Directorios de trabajo para los usuarios normales.
- **/root**.- Directorio de trabajo del *root*.
- **/lib** y **/usr/lib**.- Almacena las bibliotecas de lenguaje C/C++, estáticas y dinámicas.
- **/usr/include**.- Cabeceras para usar en programas C/C++.

- **/proc**.- Directorio virtual que crea el núcleo para mostrar información de bajo nivel a las aplicaciones sobre el estado del sistema.
- **/sbin** y **/usr/sbin**.- Almacena programas críticos de administración únicamente ejecutables por el súper usuario (*root*).
- **/usr/src**.- Código fuente del núcleo del S.O.

Es interesante echarle un vistazo al contenido de `bin`. Ahí podemos ver algunos de los programas ejecutables que usaremos normalmente. Existen órdenes como `cd` que no aparecen como programas en ninguna parte; se trata de órdenes internas del intérprete de órdenes.

La orden `ls`, al igual que otras órdenes que admitan un número variable de archivos de entrada, permite el uso de unos caracteres especiales llamados *comodines* para la creación de nombres con un patrón básico de caracteres. Los más utilizados son:

- La interrogación **?**.- Indica que en esa posición se considere cualquier carácter.
- El asterisco *****.- Indica que en esa posición se admiten cero o más caracteres.
- Caracteres entre corchetes **[**abcde3456**]**.-Indica que en esa posición debe existir uno de los caracteres que se indican en el interior de los corchetes.

A continuación se muestran algunos ejemplos:

```
$ ls /bin/b* # Lista todos los archivos que hay en /bin cuyos nombres empiezan por b
/bin/banner    /bin/bdftops  /bin/bzcat      /bin/bzgrep
/bin/basename  /bin/bibtex   /bin/bzcmp      /bin/bzip2
/bin/bash      /bin/bison    /bin/bzdiff     /bin/bzip2recover
/bin/bashbug   /bin/bunzip2  /bin/bzegrep    /bin/bzless
/bin/bc        /bin/byacc    /bin/bzfgrep    /bin/bzmore
$ ls /bin/c?e* # Nombres cuyos primer y tercer carácter sean 'c' y 'e'
/bin/checkgid.exe  /bin/clearn.exe  /bin/createlang
/bin/clear.exe     /bin/createdb    /bin/createuser
$ ls /lib/*png* # Nombres que contengan la cadena "png"
/lib/libpng.a      /lib/libpng10.a      /lib/libpng12.a
/lib/libpng.dll.a  /lib/libpng10.dll.a  /lib/libpng12.dll.a
$ ls /bin/???? # Nombres que tengan exactamente 4 caracteres
/bin/c2ph  /bin/grog  /bin/gslj  /bin/h2ph  /bin/over  /bin/view
/bin/cpan  /bin/gsbj  /bin/gslp  /bin/h2xs  /bin/ptar  /bin/zcmp
/bin/flea  /bin/gsdj  /bin/gsnd  /bin/neqn  /bin/rvim  /bin/znew
$ ls /usr/include/*[125]*.h # Nombres que contengan 1, 2, ó 5, y acaben en .h
/usr/include/db_185.h  /usr/include/ieee754.h  /usr/include/orbit-idl2.h
```

Podemos además beneficiarnos de algunas de las comodidades que ofrecen los intérpretes de órdenes, como es el *histórico de órdenes*. Este se utiliza para acceder rápidamente a alguno de los últimos órdenes introducidos sin necesidad de teclearlos de nuevo. Mediante las flechas del cursor de arriba y abajo, podemos recuperarlos.

Si escribimos los primeros caracteres bien sea de una orden o bien de un argumento de tipo archivo, y presionamos la tecla de tabulación [TAB], la *shell* intentará completar la palabra usando el nombre de algún archivo que encuentre. En el caso de que existan varias posibilidades para completar el nombre de archivo, una segunda presión mostraría las opciones posibles.

mkdir — Make Directory

Se utiliza para crear archivos de tipo directorio. Esta clase de archivos almacenan los nombres y las referencias a los archivos en disco que contienen. Como veremos más adelante, se puede referenciar más de una vez al mismo archivo en el mismo o en distintos directorios, pudiendo tener diferentes nombres en cada ubicación.

A continuación se utiliza `mkdir` para crear el directorio `Laboratorio` dentro del directorio del usuario, para crear el directorio `MisProgramas` en `/bin`, y para crear el directorio `otro` colgando del directorio padre del actual:

```
$ mkdir ~/Laboratorio
$ mkdir /bin/MisProgramas
$ mkdir ../otro
```

rmdir — Remove Directory

Elimina directorios vacíos. El argumento o argumentos deben ser nombres de directorios. Esta orden puede eliminar varios directorios si se especifican sus nombres uno detrás de otro separados por al menos un espacio. Esta orden también acepta el uso de comodines en la especificación de los nombres de directorios.

```
$ rmdir ~/Laboratorio
$ rmdir /bin/MisProgramas
$ rmdir ../otro
```

rm — Remove

Elimina archivos.

La siguiente orden elimina el archivo notas.txt que se encuentra en el directorio actual.

```
$ rm notas.txt
```

La siguiente orden elimina del directorio `/tmp` todos los archivos cuyos nombres tengan más de 3 caracteres:

```
$ rm /tmp/????*
```

Realmente `rm` tan sólo decrementa el número de referencias que se le hacen a los archivos. La eliminación se hará efectiva cuando ese número llegue a 0.

`rm` también puede eliminar archivos de tipo directorio, siendo a menudo más utilizado que `rmdir` para ese fin. Mediante el modificador `r` se borrarían recursivamente los contenidos que pudiera contener, lo que puede ser de gran ayuda. Para solicitarle un funcionamiento silencioso se usa el modificador `f`.

```
$ rm -rf GNUstep
```

Debemos considerar la peligrosidad del uso inadecuado de órdenes como estas, sobre todo siendo súper usuario.

cp — Copy

Copia archivos. Necesita al menos dos argumentos: el archivo origen, y el destino. Si el destino es el nombre de un directorio existente, creará una copia del archivo dentro del directorio conservando el nombre original.

```
$ ls
GNUstep  notas.txt  startx
$ cp notas.txt calificaciones.txt
$ ls
GNUstep  calificaciones.txt  notas.txt  startx
$ cp notas.txt /etc # Copia notas al directorio /etc
```

El nuevo archivo hereda los permisos del original, así como el usuario y el grupo.

mv — Move

Mueve archivos de una ubicación a otra en el sistema de directorios. También puede ser utilizado para renombrar el archivo si se especifica el nuevo nombre en el segundo parámetro. A diferencia de `cp`, esta orden elimina la ruta original del sistema de archivos.

```
$ mv notas.txt ..  # Mueve notas.txt al directorio padre del actual
$ ls -l ..         # Muestra el directorio padre
total 1
drwxr-xr-x    4 David    Ninguno         0 Aug  7 20:23 David
-rw-r--r--    1 David    Ninguno        21 Aug  7 01:22 notas.txt
$ mv /home/notas.txt ~ # Mueve el archivo al directorio de conexión
$ mv calificaciones.txt calificaciones_temporales.txt # Renombra el archivo
```

cat — Concatenate

Concatena o une los archivos que se le pasen como argumentos, y los saca por la salida estándar. Si no se especifican archivos para concatenar, toma los datos de la entrada estándar.

En la práctica, el uso más habitual de `cat` es mostrar el contenido de un archivo de texto.

```
$ cat notas.txt   # Mostrar el contenido del archivo notas.txt
Manuel      8,5
Miguel      6,0
Ana         8,0
```

more — More

Similar a la orden `cat`, `more` muestra por pantalla el contenido de un archivo que se especifique como argumento. En el caso de no proporcionarlo, se mostrará lo que se le pasa por la entrada estándar. La principal diferencia con `cat` es que si el archivo ocupa más de una pantalla de texto, `more` detendrá la salida de información hasta que el usuario pulse [Enter], con lo que avanza una línea, [Espacio] para avanzar hasta llenar otra pantalla, o la tecla [Q] que detiene la ejecución.

```
$ more notas.txt   # Mostrar notas.txt detenidamente
Manuel      8,5
Miguel      6,0
Ana         8,0
```

Tanto `cat` como more intentan visualizar el contenido entero del archivo de entrada. En el caso de que se trate de un archivo grande es posible que nos sea suficiente con ver únicamente algunas líneas del principio o del final del mismo. Para ello utilizaremos las órdenes `head` y `tail`:

```
$ head listín.txt   # Mostrar las primeras líneas del archivo listín.txt
...
$ tail listín.txt   # Mostrar las últimas líneas del archivo listín.txt
...
```

ln — Link

Crea un *enlace simbólico* a un archivo. Los sistemas de archivos compatibles POSIX dan soporte para un tipo especial de archivo llamado enlace simbólico, mediante el que podemos referirnos al archivo al que apunta. Se soportan dos tipos de enlaces: los fuertes y los débiles.

Un enlace fuerte a un archivo es una entrada en algún directorio que referencia a un archivo. Cada archivo tiene por lo menos un enlace fuerte, pero puede tener más.

Suponiendo que exista un archivo llamado `notas.txt`, vamos a crear un sinónimo del archivo:

```
$ ls -l
-rw-rw-r--   1 david    david         452 Aug  8 00:53 notas.txt
$ ln notas.txt notas2000.txt
$ ls -l
-rw-rw-r--   2 david    david         452 Aug  8 00:55 notas2000.txt
-rw-rw-r--   2 david    david         452 Aug  8 00:53 notas.txt
$ cat notas2000.txt  # Idéntico resulado que: $ cat notas.txt
```

Como vemos al listar el directorio aparecen dos entradas `notas.txt` y `notas2000.txt` para referirse al mismo contenido. Observamos que el número de referencias que marca cada entrada es de 2, ya que por el momento el archivo sólo está siendo referenciado por estos dos nombres. Para eliminar el archivo del disco duro, deberíamos borrar las dos entradas. En el momento que el sistema de archivos detecte que el número de referencias al archivo es de 0, lo eliminará definitivamente.

Los enlaces fuertes se pueden crear también para otros tipos de archivo como directorios. Tienen la limitación de que han de ser creado siempre referenciando a archivos que se encuentren en el mismo sistema de ficheros (pe. la misma partición del disco duro).

El enlace débil tiene una utilidad similar, pero es almacenado como un tipo especial de archivo. Además no exige que el archivo al que referencia pertenezca al mismo sistema de ficheros ni que exista realmente. Para indicar que lo que queremos crear se trata de un enlace débil, se usa el modificador `s`. Si quisiéramos un enlace débil llamado binarios al directorio `/bin`, usaríamos la siguiente orden:

```
$ ln -s /bin binarios
$ ls -l  # El listado detallado muestra el archivo al que apuntan los enlaces
lrwxrwxrwx   1 david    david           4 Aug  8 01:13 binarios -> /bin
-rw-rw-r--   2 david    david         452 Aug  8 00:53 notas2000.txt
-rw-rw-r--   2 david    david         452 Aug  8 00:53 notas.txt
$ cd binarios  # Podemos establecer el directorio de trabajo al enlace …
$ pwd          # … recién creado como si se tratara del propio directorio /bin
/home/david/binarios
$ ls ch*       # Listar todos los archivos que comiencen por ch
chgrp  chmod  chown
```

chmod — Change Mode

Cambia los permisos de acceso al archivo.

En su forma más simple la orden añade un permiso o lo elimina anteponiendo un + o un − al tipo de permiso a saber: `r`, `w`, `x`, correspondientes a lectura, escritura y ejecución respectivamente.

```
$ chmod +x guiones.sh  # Activa el atributo de ejecutable al archivo guiones.sh
$ chmod -wx lista.txt  # Hace que no se pueda modificar ni ejecutar el archivo
```

Por defecto se modifican los atributos de usuario de los archivos afectados. Para especificar explícitamente a quiénes se les aplican los cambios, se pueden usar los modificadores `u`, `g`, y `a` correspondientes al usuario, grupo, y todos respectivamente.

```
$ chmod ga+r lista.txt # Hace que el archivo sea legible por el grupo y por todos
```

Observar que en los directorios del sistema los usuarios normales no tienen permiso `w`, por lo que no pueden ser modificados. Además los directorios pertenecientes a otros usuarios no suelen tener permiso siquiera de lectura para otros usuarios.

man — Manual

El sistema de ayuda básico en el mundo de GNU/Linux está compuesto por unas páginas de manual que muchos programas suelen incorporar. Ofrecen una información más extensa que la proporcionada por la opción `--help`. El programa `man` se encarga de visualizar dichas páginas por consola. Podemos desplazarnos con las teclas del cursor arriba y abajo. Para salir, pulsaremos la tecla [Q].

```
$ man ls    # Muestra el manual de ls
$ man man # Muestra el manual de man
```

En ocasiones existe más de un tema distinto con el mismo nombre en las páginas de `man`. Para mostrarlas todas, usamos la orden `whatis`. Si lo que queremos es buscar por los contenidos de los manuales alguna coincidencia con una palabra en concreto, usaremos `apropos`. Previo al uso de cualquiera de estas dos, es necesario haber construido una base de datos con un índice de los manuales con ayuda de la orden `/usr/sbin/makewhatis`, ejecutable únicamente si se tiene privilegios de *root*.

```
# /usr/sbin/makewhatis # La base de datos se crea una sola vez
...
$ whatis sleep  # Buscar todas las entradas de manual que se llamen sleep
sleep (1)            - delay for a specified amount of time
sleep (3)            - Sleep for the specified number of seconds
$ apropos sleep # Buscar todas las entradas de manual que se contengan sleep
Tcl_Sleep (3)        - delay execution for a given number of milliseconds
apmsleep (1)         - go into suspend or standby mode and wake-up later
nanosleep (2)        - pause execution for a specified time
sleep (1)            - delay for a specified amount of time
sleep (3)            - Sleep for the specified number of seconds
usleep (1)           - sleep some number of microseconds
usleep (3)           - suspend execution for interval of microseconds
Tcl_Sleep (3)        - delay execution for a given number of milliseconds
apmsleep (1)         - go into suspend or standby mode and wake-up later
nanosleep (2)        - pause execution for a specified time
sleep (1)            - delay for a specified amount of time
sleep (3)            - Sleep for the specified number of seconds
usleep (1)           - sleep some number of microseconds
usleep (3)           - suspend execution for interval of microseconds
```

Los números que se acompañan entre paréntesis sirven para identificar cada uno de los temas en los que están divididos los manuales. Podemos usarlos para indicarle a `man` cuál es exactamente el tema de ayuda que nos interesa:

```
$ man 1 sleep  # Muestra información sobre el programa sleep
...
$ man 3 sleep  # Muestra información sobre la llamada de lenguaje C sleep(…)
...
```

SU — Switch User

Cambia la identidad del usuario actual. El argumento suele ser el nombre de *login* de usuario. Si no se introduce nombre se considera que se desea adquirir la identidad del *root*. La orden pide la clave para verificar la autenticidad del nuevo usuario.

```
$ su oscar   # Solicita un cambio de identidad para convertirnos en oscar
Password:
# whoami     # Pedir información sobre el usuario actual
oscar
```

Para regresar al usuario anterior, se introduce la orden `exit`, tal y como muestra la siguiente sesión:

```
$ su          # Se quiere ser root
Password:
# whoami      # Pedir información sobre el usuario actual
root
#             # Después de introducir algunas órdenes adicionales...
# exit        # Abandonar la identidad de root
$ whoami
David
```

shutdown

Apaga el sistema. Como suele ocurrir en los SS.OO. de envergadura, para mantener la coherencia de datos no se pueden apagar ni resetear directamente, sino que el propio sistema debe ser informado de tales intenciones. Así se tiene oportunidad de salvar información de configuración y actualizar los discos duros con los datos aún presentes en caché.

La siguiente orden pide al sistema un rearranque con la máxima brevedad posible:

```
# shutdown –r now  # Equivalente a pulsar [Ctrl]+[Alt]+[Supr] en ciertas config.
```

El parámetro `r` le indica que se desea reiniciar la máquina. También se le puede solicitar que la apague con `h`, y en algunas versiones, que la suspenda, o que la lleve a hibernación. Estos modificadores sólo funcionarán correctamente si el núcleo está correctamente configurado y el equipo lo permite.

El parámetro `now` le indica que inicie el proceso en ese mismo momento. En lugar de `now` se puede especificar un instante de tiempo absoluto, o una cantidad de minutos que deseamos que transcurran hasta que se inicie el apagado anteponiendo el signo +. En estos casos es posible que se desee transmitir un mensaje a todos los usuarios conectados para informarles de que terminen con lo que estuvieran haciendo:

```
# shutdown –h +10 "El equipo se apagará por motivos administrativos en pocos
minutos."
```

A los 10 minutos de haber introducido esta orden el sistema se apagará. Si el núcleo tiene control sobre la fuente de alimentación, esta será desconectada.

En la mayoría de los sistemas multiusuario, sólo *root* tiene permiso para ejecutar `shutdown`.

diff

El comando diff nos permite comparar dos ficheros linea a linea y nos informa de las diferencias entre ambos ficheros. Necesita al menos dos argumentos: el archivo origen, y el destino. Diff tiene muchas opciones.

Así, si queremos comparar dos ficheros, ignorando los espacios en blanco, utilizaremos el parámetro –w. Si lo que queremos es que no nos muestre las diferencias, sino que tan sólo nos informe de si son diferentes o no se usa el modificado –q. Tambien podemos solicitar que nos muestre la salida con las diferencias marcadas a dos columnas. También podemos utilizar la opción -i, que ignora la diferencia entre mayúsculas y minúsculas.

grep

Su funcionalidad es la de escribir en salida estándar aquellas líneas que concuerden con un patrón. Su sintaxis es como sigue:

```
grep [opciones] PATRÓN [ARCHIVO...]
grep [opciones] [-e PATRÓN | -f ARCHIVO] [ARCHIVO...]
```

Este comando realiza una búsqueda en los ARCHIVOs (o en la entrada estándar, si no se especifica ninguno) para encontrar líneas que concuerden con PATRÓN. Por defecto **grep** imprime en pantalla dichas líneas. Sus opciones más interesantes son:

-c

Modificar la salida normal del programa, en lugar de imprimir por salida estándar las líneas coincidentes, imprime la cantidad de líneas que coincidieron en cada archivo.

-e PATRÓN

Usar PATRÓN como el patrón de búsqueda, muy útil para proteger aquellos patrones de búsqueda que comienzan con el signo «-».

-f ARCHIVO

Obtenee los patrones del archivo ARCHIVO

-H

Imprimir el nombre del archivo con cada coincidencia.

-r

Buscar recursivamente dentro de todos los subdirectorios del directorio actual.

El patrón de búsqueda normalmente es una palabra o una parte de una palabra. También se pueden utilizar *expresiones regulares*, para realizar búsquedas más flexibles.

which

El comando which nos sirve para averiguar donde se encuentra instalado un determinado programa.

Por ejemplo, si ejecutamos:
$ which find

Nos devolverá:
/usr/bin/find

Ésto nos mostrará la primera aparición del programa buscado. Si queremos que nos muestre todas las ocurrencias que encuentre, utilizaremos el parámetro **-a**.

Así, si ejecutamos:
$ which -a find

Nos devolverá todas las ocurrencias que encuentre:
/usr/bin/find
/usr/bin/X11/find

Para realizar la búsqueda **which** localiza los ficheros ejecutables mediante el PATH.

Ésto puede sernos útil, por ejemplo, para comprobar si tenemos dos versiones de un mismo programa instalado en diferentes directorios.

gzip

Los ficheros comprimidos utilizan menos espacio en el disco y se descargan más rápido que los ficheros no comprimidos. Puede comprimir ficheros Linux con la herramienta de compresión open-source Gzip o on Zip, reconocida por la mayoría de sistemas operativos.

Por convención, a los ficheros comprimidos se les da la extensión .gz. El comando **Gzip** crea un fichero comprimido que finaliza con .gz; **Gunzip** extrae los ficheros comprimidos y suprime el fichero .gz.

Para comprimir un fichero, teclee el siguiente comando en el indicador de comandos de la shell:

$ gzip filename.ext

El fichero será comprimido y guardado como filename.ext.gz**.**

gunzip

Realiza la descompresión (o expansión) de un fichero previamente comprimido mediante la orden gzip. Para expandir un fichero comprimido, teclee:

gunzip filename.ext.gz

El filename.ext.gz se borra y se reemplaza con filename.ext

tar

Los ficheros tar ubican diferentes ficheros o los contenidos de un directorio o directorios en un fichero. Este es un buen modo de crear copias de seguridad y archivos. Habitualmente, los ficheros tar acaban con la extensión .tar.
Para crear un fichero tar, teclee:

tar -cvf filename.tar files/directories

En este ejemplo, filename.tar representa el fichero que está creando y files/directories representa los ficheros o directorios que quiere introducir en el nuevo fichero.

Puede utilizar nombres de rutas absolutos o relativos para estos ficheros y directorios . Separe los nombres de los ficheros y directorios con un espacio.

La siguiente entrada creará un fichero tar usando un nombre de ruta absoluto:

tar -cvf foo.tar /home/mine/work /home/mine/school

El comando anterior situará todos los ficheros en el subdirectorio /work y el subdirectorio /school en un nuevo fichero llamado foo.tar en el directorio donde está trabajando.

El comando **tar -cvf foo.tar file1.txt file2.txt file3.txt** situará file1.txt, file2.txt y file3.txt en un nuevo fichero llamado foo.tar.

Para listar los contenidos de un fichero tar, teclee:
tar -tvf foo.tar

Para extraer los contenidos de un fichero tar, teclee:
tar -xvf foo.tar

Este comando no suprime el fichero .tar file, pero ubica copias del contenido de .tar en el directorio en el que está trabajando actualmente.

El comando **tar** no coomprime ficheros automáticamente. Puede comprimir ficheros tar con:
tar -czvf foo.tar

A los ficheros tar se les da generalmente la extensión .tgz y son comprimidos con gzip.

Para expandir un fichero tar comprimido escriba:
tar -xzvf foo.tgz

Conceptos adicionales sobre el sistema

Hasta aquí hemos visto algunos de las órdenes más básicos del sistema. Ahora vamos a hacer un pequeño alto para comentar algunas de las posibilidades que nos brinda el sistema.

Muchas de las ordenes de manejo del sistema como hemos visto utilizan la entrada estándar –el teclado– para adquirir datos del exterior, y la salida estándar –el monitor– para mostrarlos al usuario. Podría ser que nos interesara alterar este comportamiento. Para ello vamos a emplear los operadores de redirección y las tuberías.

Redireccionamiento de la entrada/salida estándar

El intérprete de órdenes nos permite utilizar unos operadores especiales que permiten que los flujos estándar sean tomados o escritos desde o hacia un archivo. Por ejemplo, para almacenar un listado de archivos del directorio `/bin`, usaríamos el operador de Redireccionamiento de la salida estándar > como se muestra a continuación:

```
$ ls -la /bin > listado.txt          # Se lista /bin
```

En este caso la *shell* ha creado un archivo llamado `listado.txt`, sobrescribiéndolo si ya existía previamente, y lo ha rellenado con el texto que produce la orden `ls -la`. Usando el operador >> obtenemos igualmente una redirección de la salida hacia un archivo, pero añadiéndole la información al final, no sobrescribiéndolo.

```
$ ls -la /usr/bin >> listado.txt     # Se le añade el listado de /usr/bin
```

Es posible también redireccionar la salida estándar de error, por medio de los operadores `<&` o `&>` conseguimos que todas las dos salidas estándar queden redireccionadas. Para redireccionar sólo la salida de error, usaremos el operador `2>`.

A continuación vamos a utilizar el operador de redirección de la entrada < para mostrar mediante more el contenido del archivo recién generado.

```
$ more < listado.txt
```

Similarmente a como ocurría antes, la *shell* está encargada de abrir el archivo listado.txt para proporcionárselo a `more` como entrada.

A menudo, cuando se necesita que la salida estándar que genera una aplicación se convierta en la entrada estándar de otra aplicación, no se desea dar los pasos intermedios para crear un archivo de datos temporal para luego pasárselo a la siguiente aplicación. Esto es aún más acusado en el caso de que la aplicación que genera datos o la que los recibe, empleen mucho tiempo en terminar de ejecutarse. Es por ello que el sistema nos proporciona el operador de tubería `|` que podemos encontrar en los teclados españoles en

la tecla [1]. Al separar dos órdenes por este carácter, conseguimos que el sistema las ejecute en paralelo redirigiendo automáticamente la salida estándar de la primera a la entrada estándar de la segunda. En el caso de que la aplicación que reciba los datos de entrada realice un procesamiento sobre los mismos, y los reenvíe Se suele decir que la segunda aplicación *filtra* la información generada por la primera.

A continuación se muestra el listado del directorio `/bin` a través de more usando una tubería:

```
$ ls -la /bin | more
```

Tanto los operadores de redirección como el de tubería pueden ser combinados para formar expresiones más complejas. En el siguiente ejemplo antes de pasar los datos a `more`, son filtrados por la orden `sort`, que realiza una reordenación alfabética de los mismos:

```
$ ls /bin | sort -r | more
```

Variables de entorno

En ocasiones se hace necesario contar con unos valores de configuración para que puedan ser consultados o modificados por las aplicaciones. Las variables de entorno son un método simple y eficaz de almacenar cadenas de caracteres vinculadas a un nombre de variable. La biblioteca estándar de C proporciona funciones específicas para poder consultarlas o establecerlas. Si quisiéramos utilizar el contenido de alguna de estas variables con el intérprete de órdenes, deberemos anteponer el carácter `$` al nombre de la misma.

A continuación se le asigna el contenido Politécnico a la nueva variable `EDIFICIO`. En el ejemplo, la orden `echo` tan solo se utiliza para mostrar un eco de lo que se ponga a continuación del mismo:

```
$ EDIFICIO=Politécnica
$ echo Escuela o facultad $EDIFICIO de la UA.
Escuela o facultad Politécnica de la UA.
```

Si quisiéramos que la cadena contuviera espacios en su interior, sería necesario entrecomillarla:

```
$ EDIFICIO="Enfermería y Fisioterapia"
$ echo Escuela o facultad $EDIFICIO de la UA.
Escuela o facultad Enfermería y Fisioterapia de la UA.
```

Las variables de entorno así definidas sólo son utilizables por la aplicación que las crea. En el caso más habitual en que queramos que esa variable sea visible por todos los programas que se creen a continuación, deberemos utilizar la palabra clave `export`.

```
$ export EDIFICIO                # Exporta la variable existente EDIFICIO
$ export CARRETERA="Nacional II" # Crea y exporta la variable CARRETERA
```

La variable de entorno más utilizada y conocida es `PATH`. Se trata de una variable que almacena varias rutas de directorios separadas por el carácter de dos puntos, que el sistema de ficheros utilizará para crear rutas absolutas a partir de rutas relativas en el momento de buscar archivos ejecutables.

```
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/david/bin
```

Como vemos en este ejemplo, `PATH` almacena varias rutas susceptibles de contener archivos binarios ejecutables. Observamos que aparece una entrada a un directorio `bin` que se encuentra en nuestro directorio de usuario. Si tal carpeta no existe, ahora es un buen momento para crearla. Sería buena idea poner ahí todos los archivos ejecutables que generemos en sucesivas prácticas, de tal forma que sean buscados en esa ubicación si es que no se encontraran en las otras rutas. Hay que tener en cuenta que para ejecutar un archivo del que no se indica su ruta absoluta, que se encuentre en el directorio de trabajo actual pero no se

encuentre en PATH, por convenio hay que anteponerle los caracteres ./ que indican que se tome como ruta de búsqueda la del propio directorio de trabajo:

```
$ cd  # Nos colocamos en el directorio de usuario (que no está en PATH)
$ cp /bin/ls listar  # Hacer una copia del programa /bin/ls
$ ./listar -a /var  # Para que la shell encuentre el programa se utiliza ./
.   arpwatch  catman  gdm  local  log  preserve  spool  tmp
..  cache     db      lib  lock   nis  run       state  yp
```

La siguiente orden redefine la variable PATH para que contenga lo que tenía antes más una ruta adicional:

```
$ export PATH="$PATH:/usr/sbin"
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/david/bin:/usr/sbin
```

Las variables de entorno se pierden cada vez que se reinicia el sistema. Si quisiéramos hacerlas persistentes podríamos incluirlas en algún archivo de configuración de recursos inicial como .bashrc.

Si se desea automatizar una secuencia de órdenes para que se ejecuten automáticamente, se pueden usar los *archivos de guiones* o de *batería de órdenes*. Se trata de archivos de texto que incorporan una lista de órdenes compatibles con un *shell* concreto, que se ejecutará cuando sea invocado. Los intérpretes de órdenes, también admiten construcciones típicas de control de flujo al igual que los lenguajes de programación. La primera línea de estos archivos debería contener el nombre del programa necesario para interpretar correctamente la semántica de las construcciones interiores, de una manera reconocible; en el caso de guiones para bash sería:

```
#! /bin/bash
```

La explicación pormenorizada de la programación de estos archivos está fuera del propósito de esta práctica. Una vez creado el archivo, es necesario otorgarle los derechos de ejecución x necesarios mediante chmod, para que el intérprete lo pueda procesar.

Nociones sobre la multitarea

Multitarea en GNU/Linux significa que se pueden tener varias tareas cargadas en memoria encargándose el sistema de ejecutarlas convenientemente según ciertas reglas internas. La multitarea es un requisito necesario para cualquier sistema de tiempo compartido en el que puede haber más de un usuario interaccionando al mismo tiempo con la máquina. Ya se ha comentado la importancia que estos sistemas dan a la protección y aislamiento entre usuarios, que se ve reflejada en un sistema de archivos robusto sobre el que se han asignado distintos privilegios y atributos. Es el núcleo el encargado de repartir los recursos del sistema, concretamente el hardware, la información, y el tiempo, de tal forma que ningún usuario no autorizado pueda monopolizar el sistema, o perjudicar significativamente a los demás.

A una máquina pueden conectarse varios usuarios a través de cualquier medio que sea capaz de mantener un terminal de comunicaciones, sobre el que se pueda construir el concepto de consola. Adicionalmente, también es posible que cada terminal pueda ser compartido por varios usuarios que estén activos al mismo tiempo cada uno con su propia consola; incluso es posible que un mismo usuario esté utilizando varias consolas simultáneamente. En el caso del terminal estándar de los ordenadores de sobremesa, compuesto esencialmente por un teclado y un monitor, podemos solicitar al sistema la habilitación de más consolas principalmente de dos maneras:

- Hallándonos en una sesión de *X-Window*, el sistema gráfico de ventanas más ampliamente utilizado en Unix, podemos solicitar la creación de terminales en ventana, como los que genera el programa xterm.

- Presionando la tecla [Alt] más una tecla de función [F1] hasta [F6], podemos acceder a una consola a pantalla completa. Para regresar a una sesión abierta de X-Window, podemos emplear la combinación [Atl] + [F7].

No todos los programas que ejecutemos tienen la rápida ejecución de los vistos hasta ahora. Las aplicaciones interactivas, y las de cálculo intensivo suelen estar cargadas durante un tiempo muy prolongado. En el caso de que sean iniciadas desde la consola, sería muy molesto tener que dejarla bloqueada hasta que el la aplicación termine. Por ello los intérpretes de órdenes permiten especificar que se desea lanzar esa tarea sin que monopolice el uso de la consola, lo que también se conoce como ejecución de fondo (en *background*), o ejecución sumergida. Esto se consigue posponiendo el símbolo `&` después de la orden que invoque a ese programa separados por al menos un espacio.

En el siguiente ejemplo se va a utilizar el programa `sleep` cuyo única finalidad es la de emplear en ejecutarse tantos segundos como se especifique en su argumento.

```
$ sleep 10 & # Si no se pusiera el &, perderíamos la consola durante 10 segundos
[1] 1039
$ . . .     # Se realizan otras acciones con la consola
 [1]+  Done                    sleep 10
```

Pasados los 10 segundos que se habían solicitado, la aplicación termina. En la siguiente oportunidad de mostrar texto, se mostrará un mensaje indicando el hecho de la finalización de la tarea. Mediante las órdenes internas `bg` y `fg`, podemos hacer que el proceso pase a segundo plano y a primer plano, respectivamente.

Vemos que inmediatamente de solicitar la ejecución de fondo, el sistema nos presenta dos números. El primero de ellos es un identificador fácil de recordar que queda vinculado a este proceso hasta que termine, momento en el que se mostrará de nuevo. El segundo número es el llamado *identificador del proceso* o *pid*. Cada proceso en Unix tiene un identificador de ese tipo lo que es de gran utilidad para realizar ciertas acciones sobre esa tarea.

Para consultar información sobre los procesos que se encuentran actualmente en el sistema, utilizamos la orden `ps`. Invocada tal cual, tan solo muestra los procesos iniciados por esa consola. Para ver todos los procesos del sistema podemos utilizar el modificador `A`:

```
$ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:05 init
    2 ?        00:00:00 kflushd
    3 ?        00:00:00 kupdate
    4 ?        00:00:00 kpiod
    5 ?        00:00:00 kswapd
  279 ?        00:00:00 portmap
  334 ?        00:00:00 syslogd
  345 ?        00:00:00 klogd
  361 ?        00:00:00 atd
  377 ?        00:00:00 crond
  397 ?        00:00:00 inetd
  413 ?        00:00:00 lpd
  444 ttyS0    00:00:00 gpm
  474 ?        00:00:00 httpd
  487 ?        00:00:00 xfs
  551 tty1     00:00:00 login
  552 tty2     00:00:00 mingetty
```

```
  553 tty3     00:00:00 mingetty
  556 tty6     00:00:00 mingetty
  908 tty1     00:00:00 bash
  956 ?        00:00:00 in.telnetd
  957 pts/1    00:00:00 login
  958 pts/1    00:00:00 bash
 1026 tty5     00:00:00 mingetty
 1027 tty4     00:00:00 mingetty
 1055 pts/1    00:00:00 ps
```

Podemos ver la gran cantidad de procesos que se encuentran activos en el sistema aun sin contar con los programas del usuario. Los que su nombre acaba con la letra `d` suele tratarse de los llamados demonios del sistema. Se trata de programas que están cargados de continuo a la espera de que se produzcan eventos en algún dispositivo –normalmente de comunicaciones-, para iniciar la ejecución de las acciones pertinentes preestablecidas.

Los procesos en Unix tienen unas relaciones de parentesco estrictas. Cada proceso es generado a su vez por otro proceso, y ambos se quedan referenciados mutuamente durante sus existencias. Al proceso generador se le llama padre, y al proceso generado se le llama hijo. Llamaremos descendencia a los procesos hijos y a los hijos de los hijos de un proceso. Si un proceso muere, mueren todos sus descendientes; esta es una opción interesante desde el punto de vista de seguimiento de recursos y coherencia del sistema. Para que un proceso muera es necesario que todos sus descendientes hayan muerto previamente. Se puede experimentar creando varios procesos desde una consola gráfica como `xterm`, y destruirla después. Cualquier proceso que hubiera quedado en funcionamiento será automáticamente destruido. Existe un proceso que es el progenitor de todos los demás (no tiene padre), llamado `init`.

En GNU/Linux se dispone de muchas formas para que los procesos intercambien información entre ellos, y por supuesto con el núcleo, algunas de las cuales ya se han visto. Uno de los mecanismos de más bajo nivel es el basado en *señales*. Las señales son eventos asíncronos que se le envían a un proceso para notificarle que ha ocurrido un cierto evento. Las señales son especialmente útiles en los S.T.R. por su similitud con las interrupciones hardware.

Para enviar señales a los procesos desde la consola, podemos utilizar la orden kill. Usando el modificador l podemos solicitarle que nos indique qué tipos de señales podemos enviar:

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGIOT       7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR
```

La señal `SIGINT` es la que se envía al pulsar [Ctrl] + [C]. Sirve para interrumpir a un proceso.

La señal `SIGTERM` se envía cuando queremos solicitar la terminación a un proceso; de esta forma tendría la oportunidad de cerrar controladamente salvando por ejemplo datos al disco. El proceso es libre de ignorar esta señal.

La señal `SIGKILL` mata forzosamente a un proceso.

En el siguiente ejemplo se mata a la aplicación que tiene el `pid` de 1095.

```
$ ps
  PID TTY          TIME CMD
  958 pts/1    00:00:00 bash
 1095 pts/1    00:00:00 vi
 1136 pts/1    00:00:00 ps
$ kill -9 1095
[1]+  Killed                  vi /usr/include/linux/signal.h
```